Основы глубинного обучения

Лекция 4

Оптимизация в глубинном обучении.

Свёрточные архитектуры.

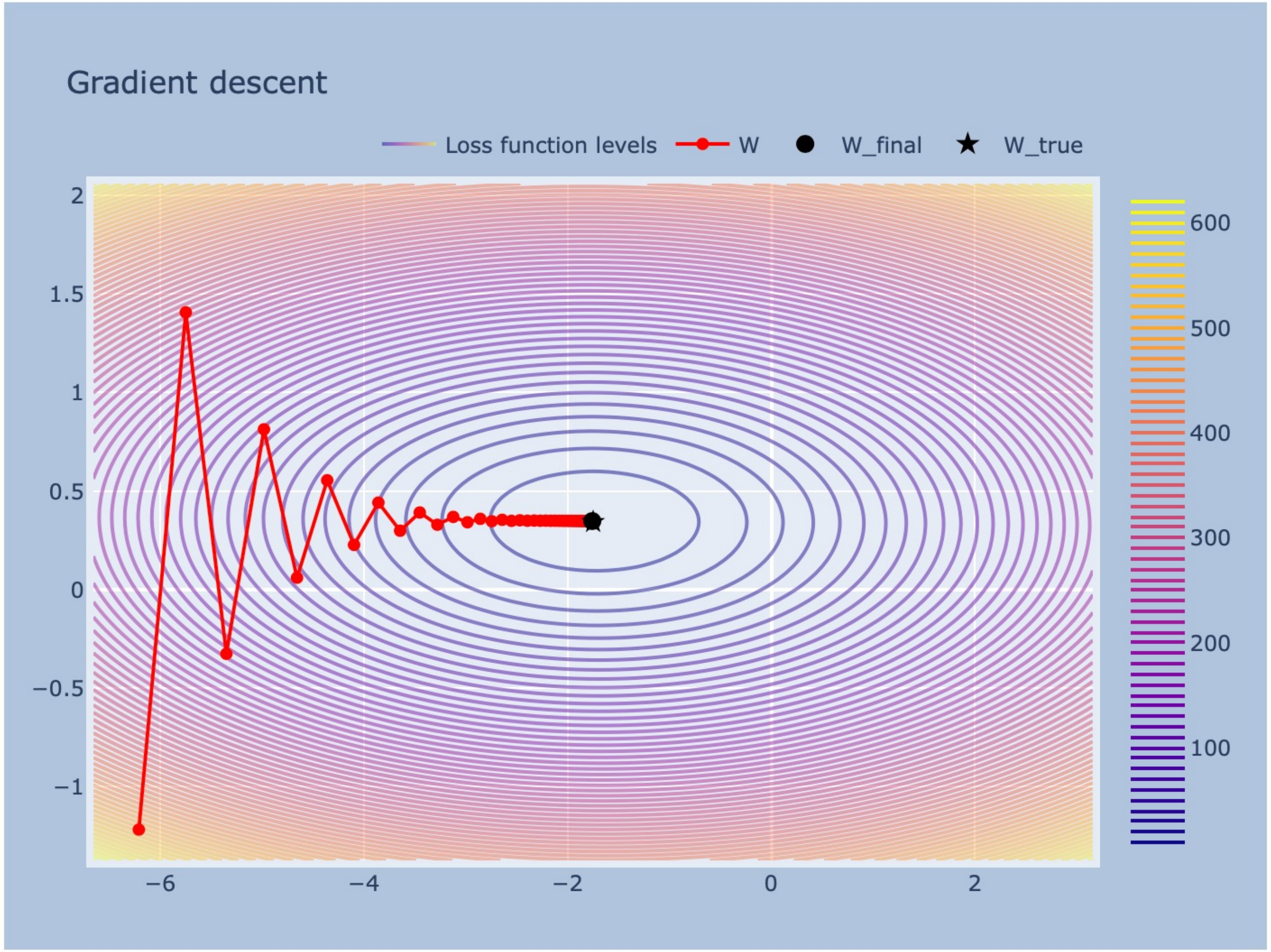
Евгений Соколов

esokolov@hse.ru

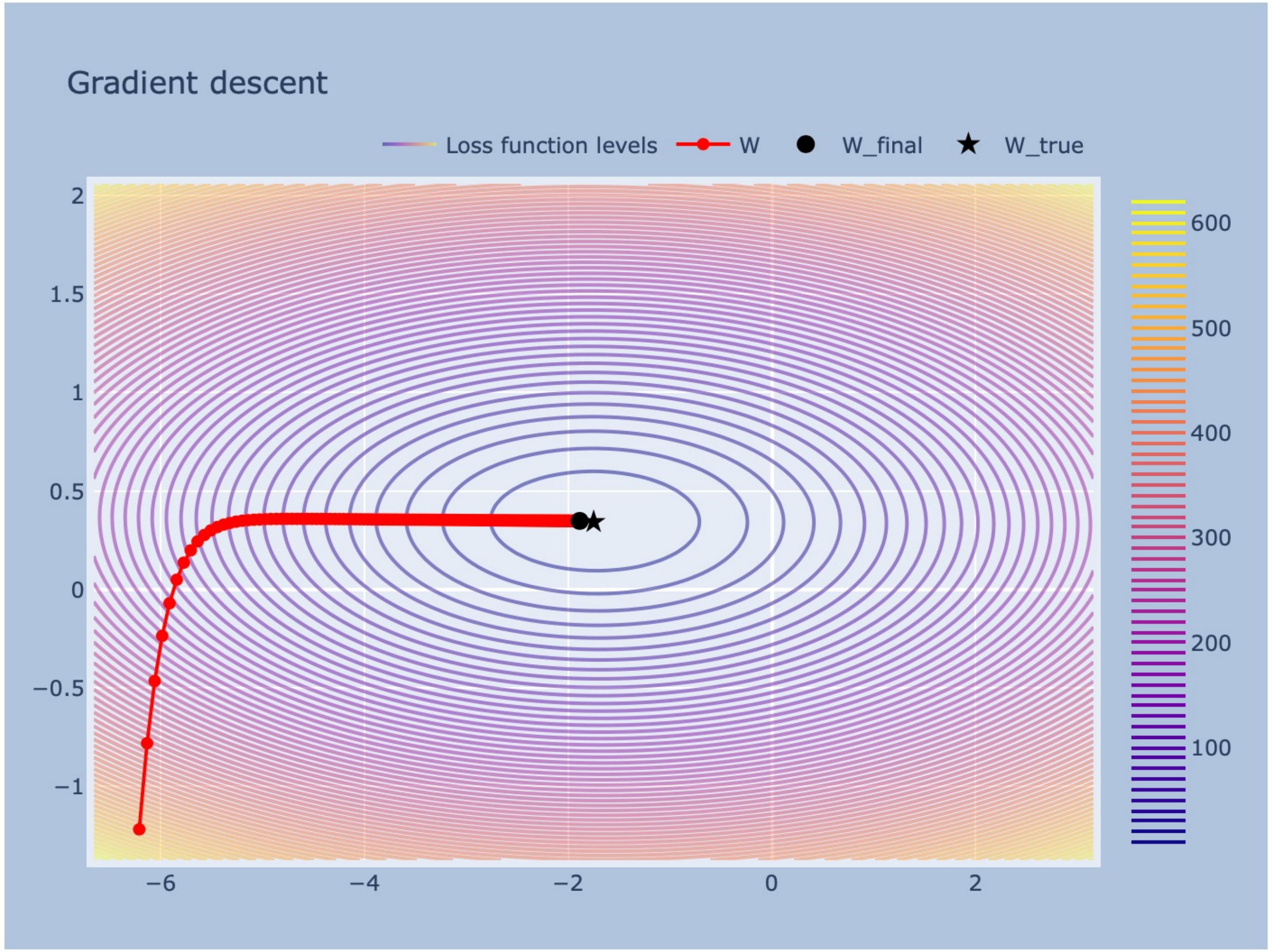
НИУ ВШЭ, 2025

Модификации градиентного спуска

Проблемы


Gradient descent
Loss function levels
W
W_final
W_true
2
1.5
1
0.5
0
−0.5
−1
−6
−4
−2
0
2
600
500
400
300
200
100

Проблемы



Gradient descent
Loss function levels
W
W_final
W_true
2
1.5
1
0.5
0
−0.5
−1
−6
−4
−2
0
2
600
500
400
300
200
100

Проблемы

- Если у функции «вытянуты» линии уровня, то градиентный спуск требует аккуратного выбора длины шага и будет долго сходиться

Momentum

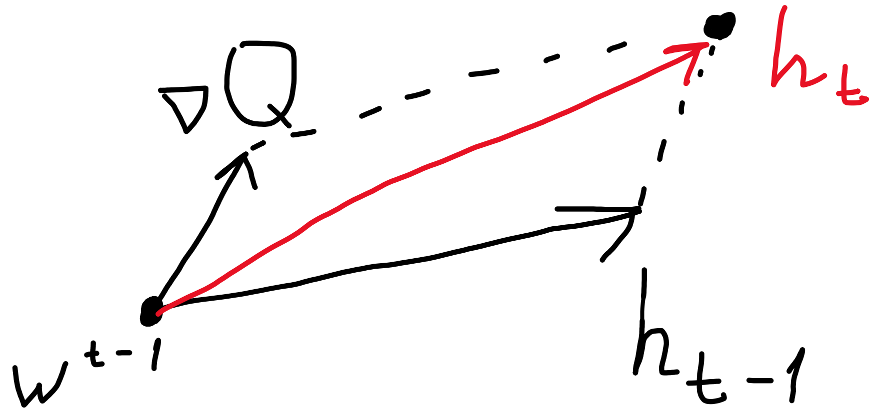
$$h_t = \alpha h_{t-1} + \eta_t \nabla Q(w^{t-1})$$
$$w^t = w^{t-1} - h_t$$

- h_t — «инерция», усреднённое направление движения
- α — параметр затухания

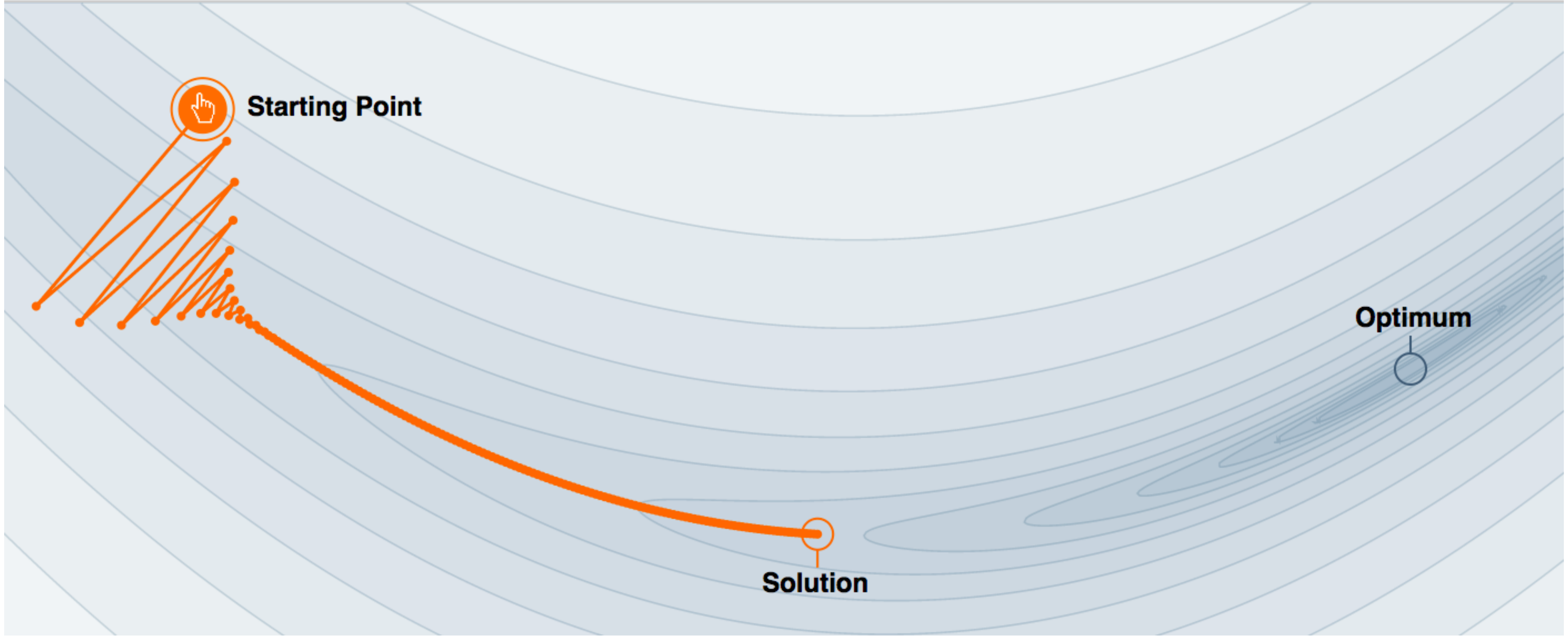
- Как будто шарик, который катится в сторону минимума, очень тяжёлый

Momentum

$$h_t = \alpha h_{t-1} + \eta_t \nabla Q(w^{t-1})$$
$$w^t = w^{t-1} - h_t$$

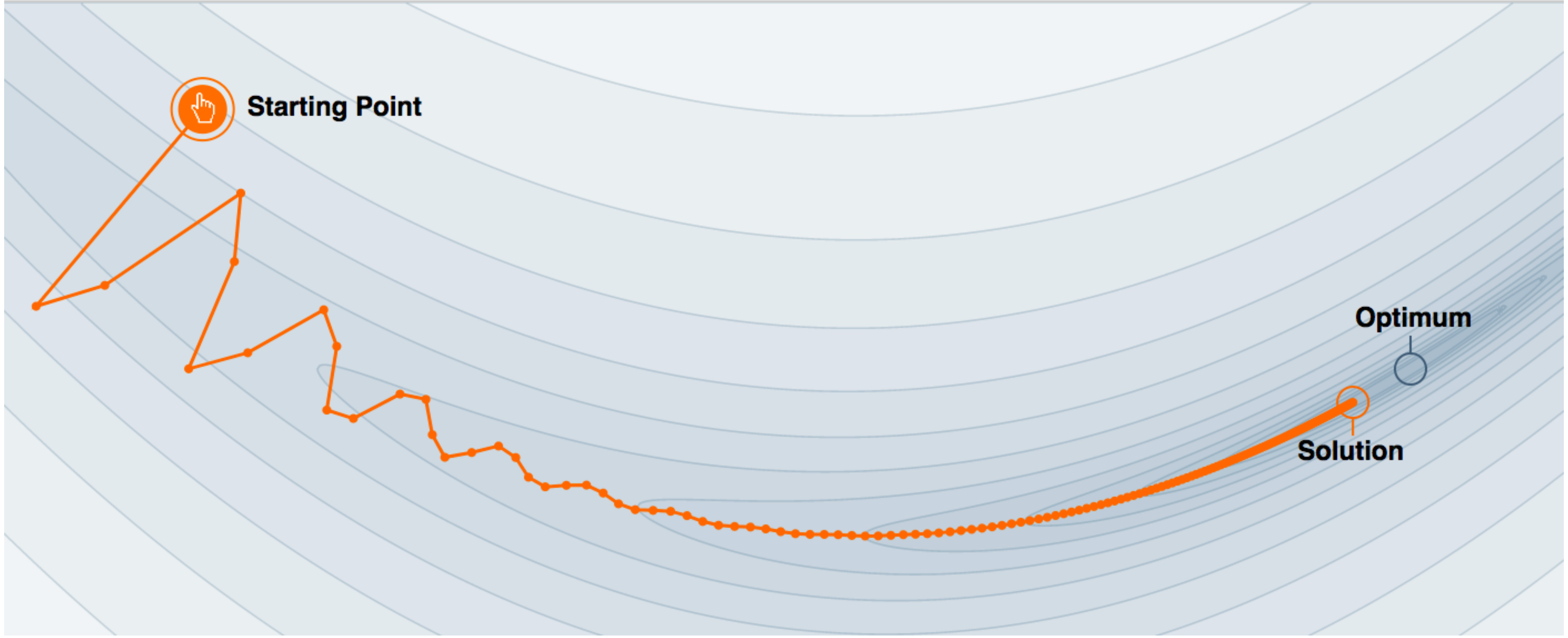


Без инерции



https://distill.pub/2017/momentum/

С инерцией

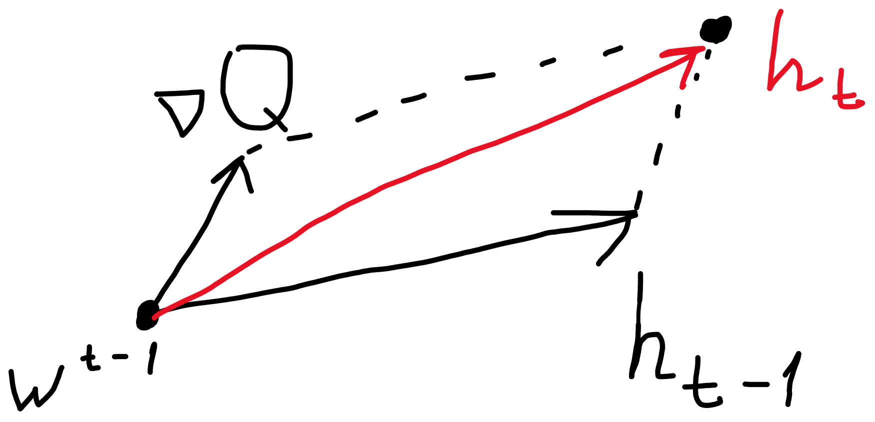


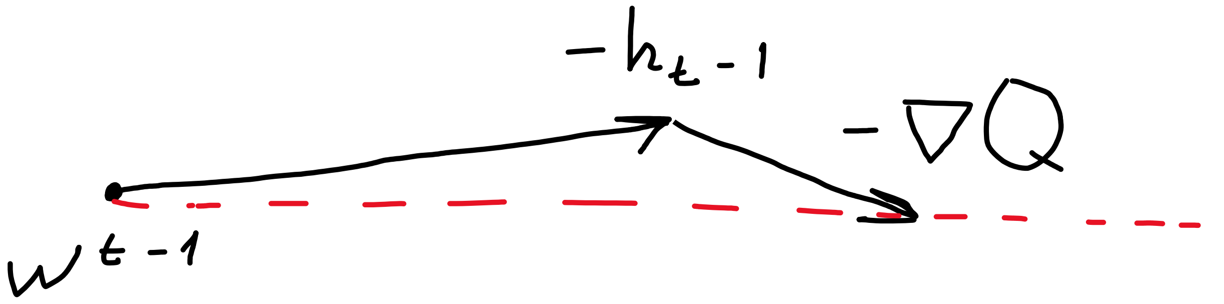
https://distill.pub/2017/momentum/

Nesterov Momentum

$$h_t = \alpha h_{t-1} + \eta_t \nabla Q(w^{t-1} - \alpha h_{t-1})$$
$$w^t = w^{t-1} - h_t$$

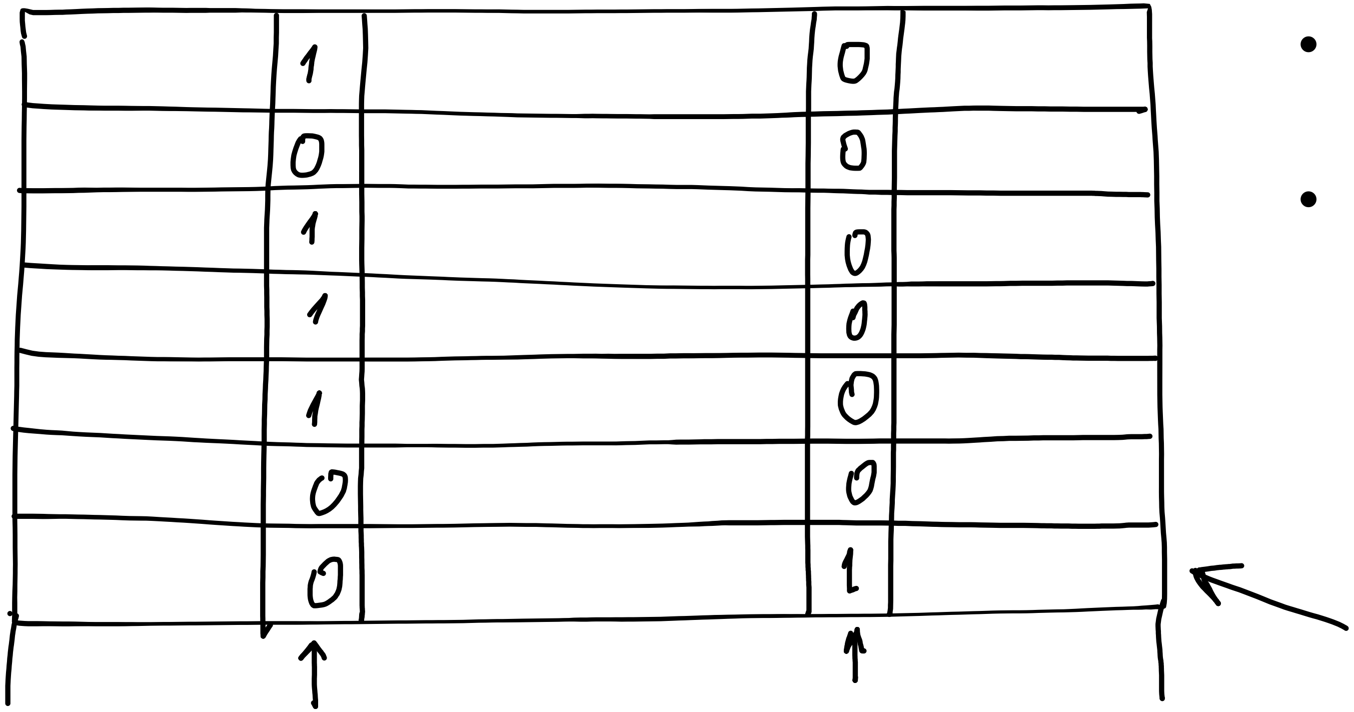
- $w^{t-1} - \alpha h_{t-1}$ — неплохая оценка того, куда мы попадём на следующем шаге





Проблема с разреженными данными

- Пример: модель над категориальными признаками
- Используем one-hot кодирование



тут уже медленные шаги

популярная категория

редкая категория

- Делаем стохастический градиентный спуск
- Через 7 шагов мы сделаем 4 обновления веса популярной категории и 1 обновление веса редкой категории

Проблема с разреженными данными

- По разным параметрам мы движемся с разной скоростью
- Будет здорово это учитывать — иначе мы обучим разные параметры с разным качеством

Проблема с разными масштабами

- Допустим, признаки имеют разный масштаб — от единиц до миллионов
- Тогда странно шагать по каждому параметру с одинаковой скоростью

AdaGrad

$$G_j^t = G_j^{t-1} + \left(\nabla Q(w^{t-1})\right)_j^2$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}} \left(\nabla Q(w^{t-1})\right)_j$$

- По каждому параметру своя скорость
- η_t можно зафиксировать
- Длина шага может убывать слишком быстро и привести к ранней остановке

RMSProp

$$G_j^t = \alpha G_j^{t-1} + (1-\alpha)\left(\nabla Q(w^{t-1})\right)_j^2$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}}\, g_{tj}$$

- α можно взять около 0.9
- Скорость зависит только от недавних шагов

Adam

$$m_j^t = \frac{\beta_1 m_j^{t-1} + (1-\beta_1)\left(\nabla Q(w^{t-1})\right)_j}{1-\beta_1^t}$$

$$v_j^t = \frac{\beta_2 v_j^{t-1} + (1-\beta_2)\left(\nabla Q(w^{t-1})\right)_j^2}{1-\beta_2^t}$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{v_j^t} + \epsilon} m_j^t$$

- Рекомендации: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

Adam

- Совмещает в себе идеи из метода инерции и RMSProp

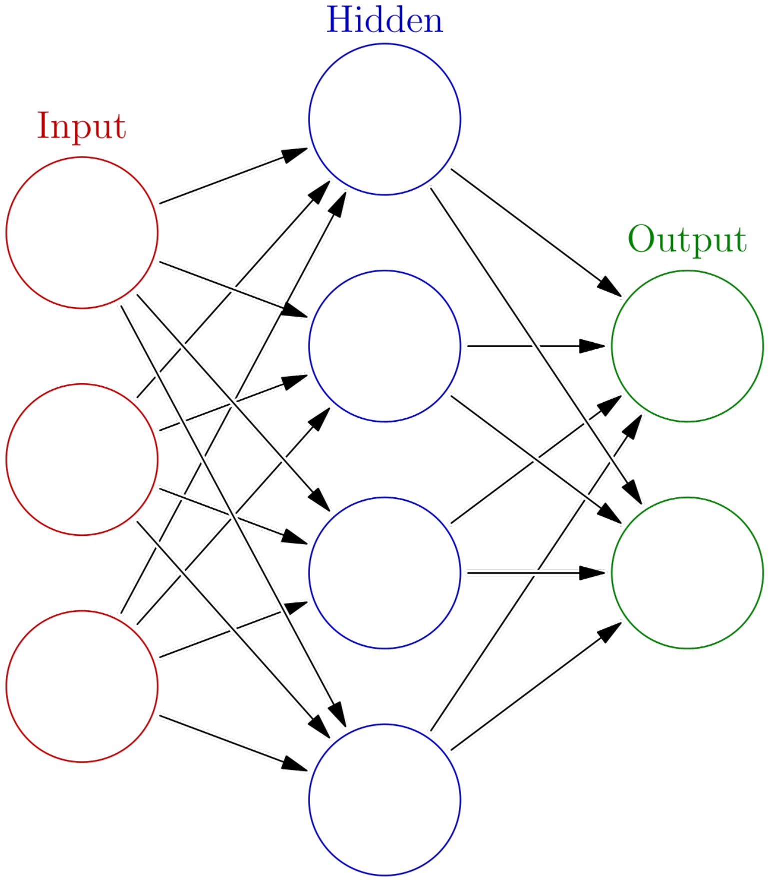
Dropout

Борьба с переобучением

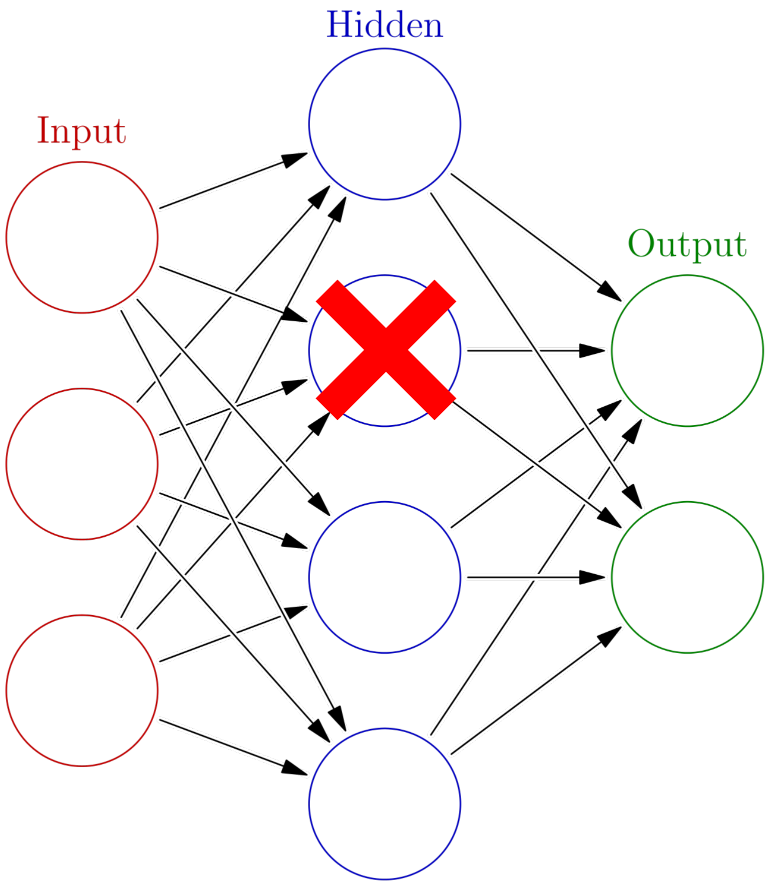
- Сокращение числа параметров (свёрточные слои помогают с этим)
- Регуляризация

- Можно как-то ещё мешать модели подгоняться под обучающую выборку

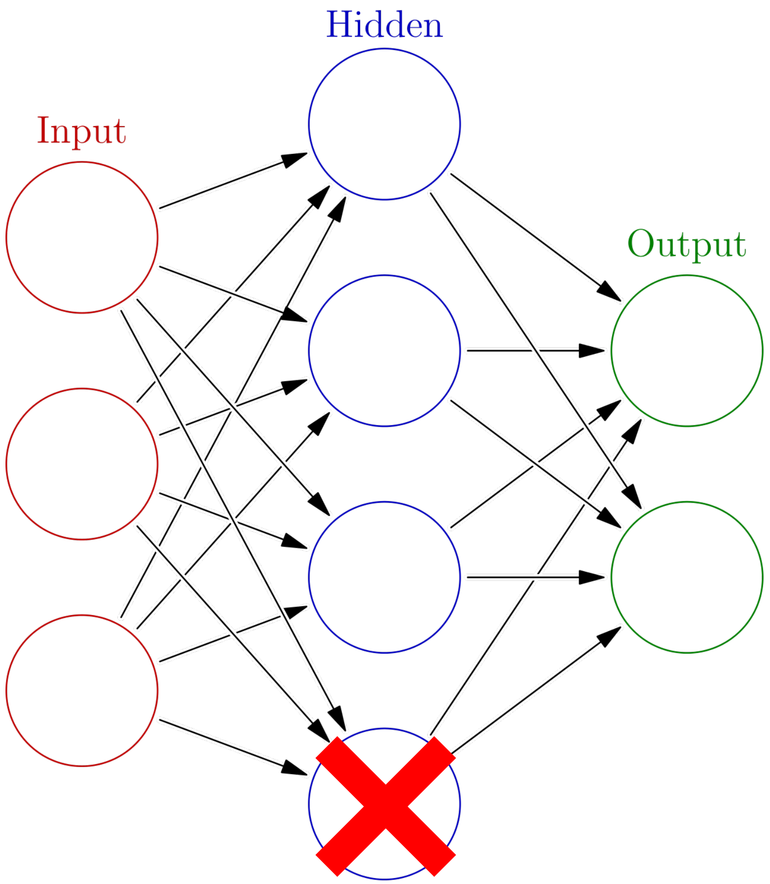
Dropout



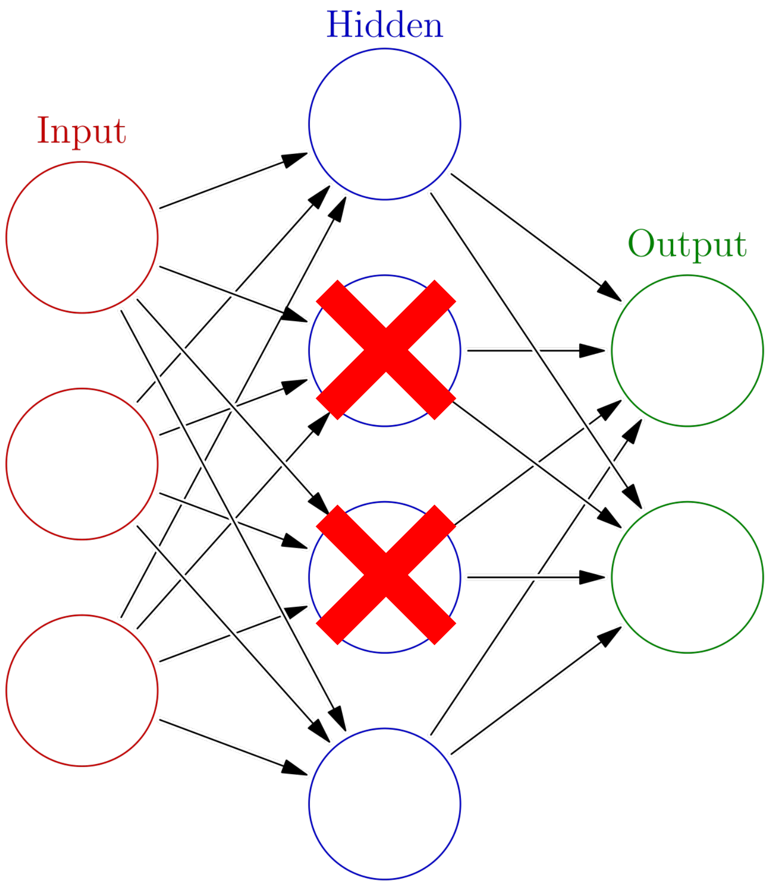
Dropout



Dropout



Dropout



Dropout

- Можно определить как слой $d(x)$
- Параметров нет, единственный гиперпараметр — p (вероятность удаления нейрона)
- На этапе обучения:

$$d(x) = \frac{1}{1-p} m \circ x$$

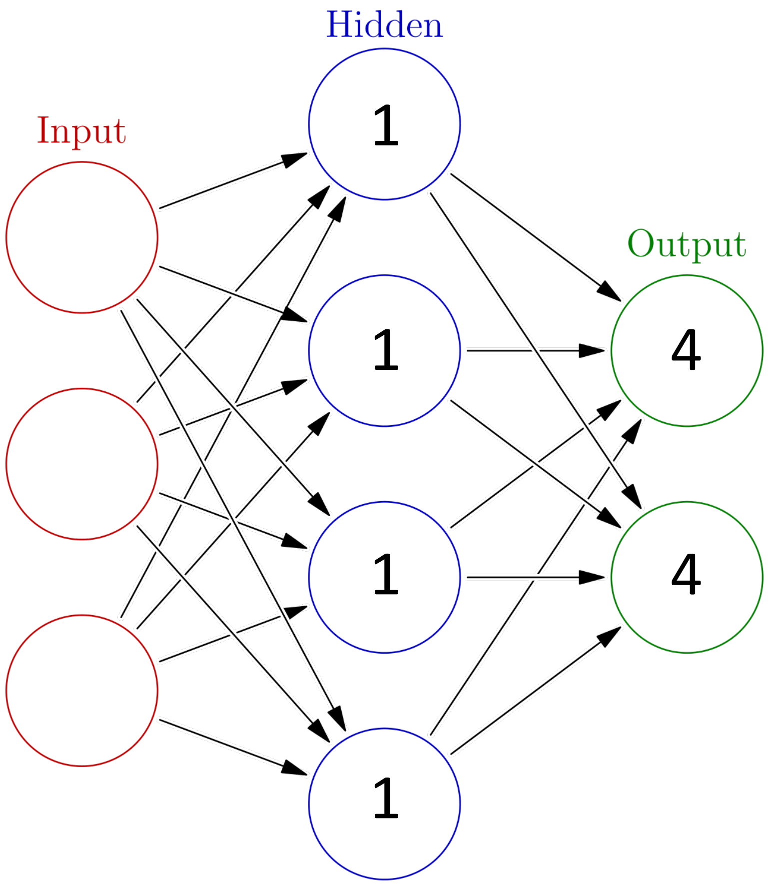
(m — вектор того же размера, что и x, элемент берутся из распределения $\mathrm{Ber}(p)$)

- Деление на p — для сохранения суммарного масштаба выходов

https://jmlr.org/papers/v15/srivastava14a.html

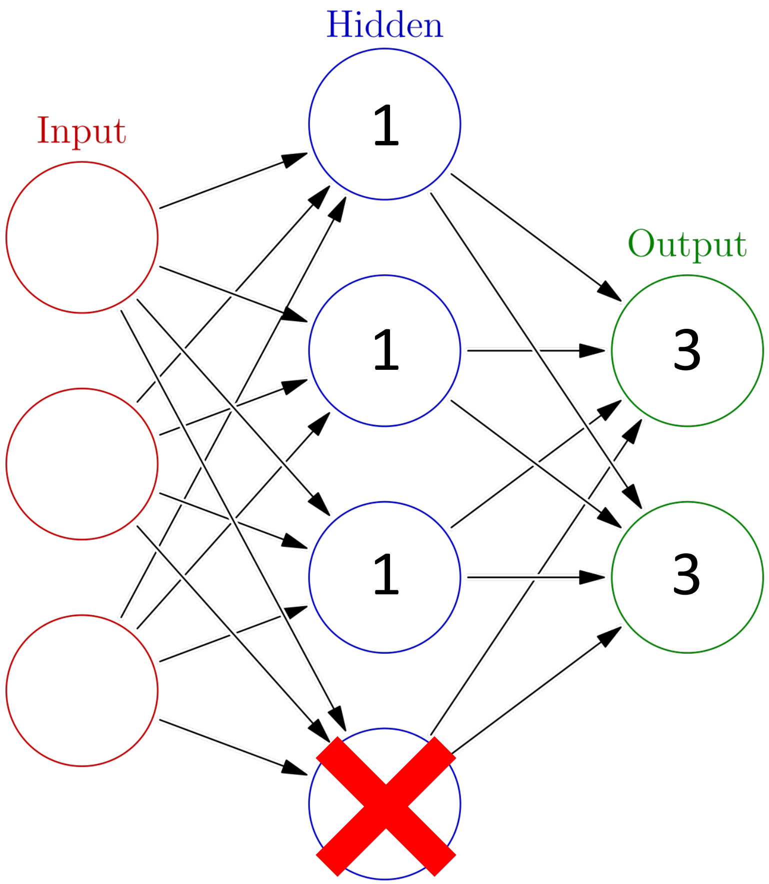
Dropout

Пусть все веса единичные



Dropout

Пусть все веса единичные



Надо компенсировать снижение масштаба суммы выходов!

Dropout

- На этапе обучения:

$$d(x) = \frac{1}{1-p} m \circ x$$

- На этапе применения:

$$d(x) = x$$

В оригинальной статье нет нормировки на этапе обучения, но есть домножение на $(1 - p)$ на этапе применения

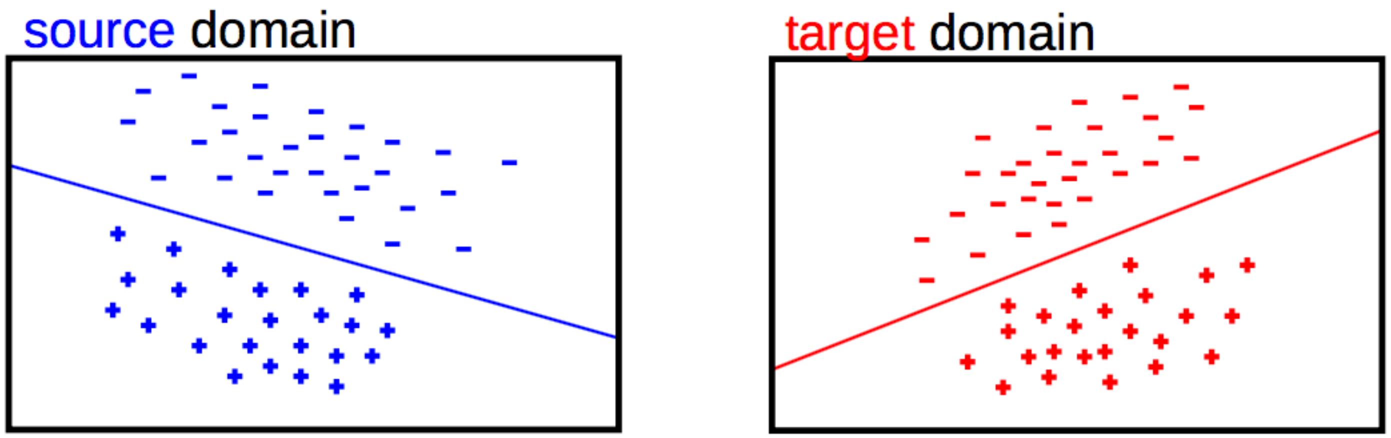
Вариант на слайде — inverted dropout (чуть меньше операций во время применения сети)

Dropout

- Интерпретация: мы обучаем все возможные архитектуры нейросетей, которые получаются из исходной выбрасыванием отдельных нейронов
- У всех этих архитектур общие веса
- На этапе применения (почти) усредняем прогнозы всех этих архитектур

Нормализации

Covariate shift



Covariate shift

- В классическом машинном обучении — изменение распределения данных
- Много методов решения

Domain adaptation

- Объекты по-разному распределены на обучении и на контроле
- Идея: взвешивать объекты при обучении

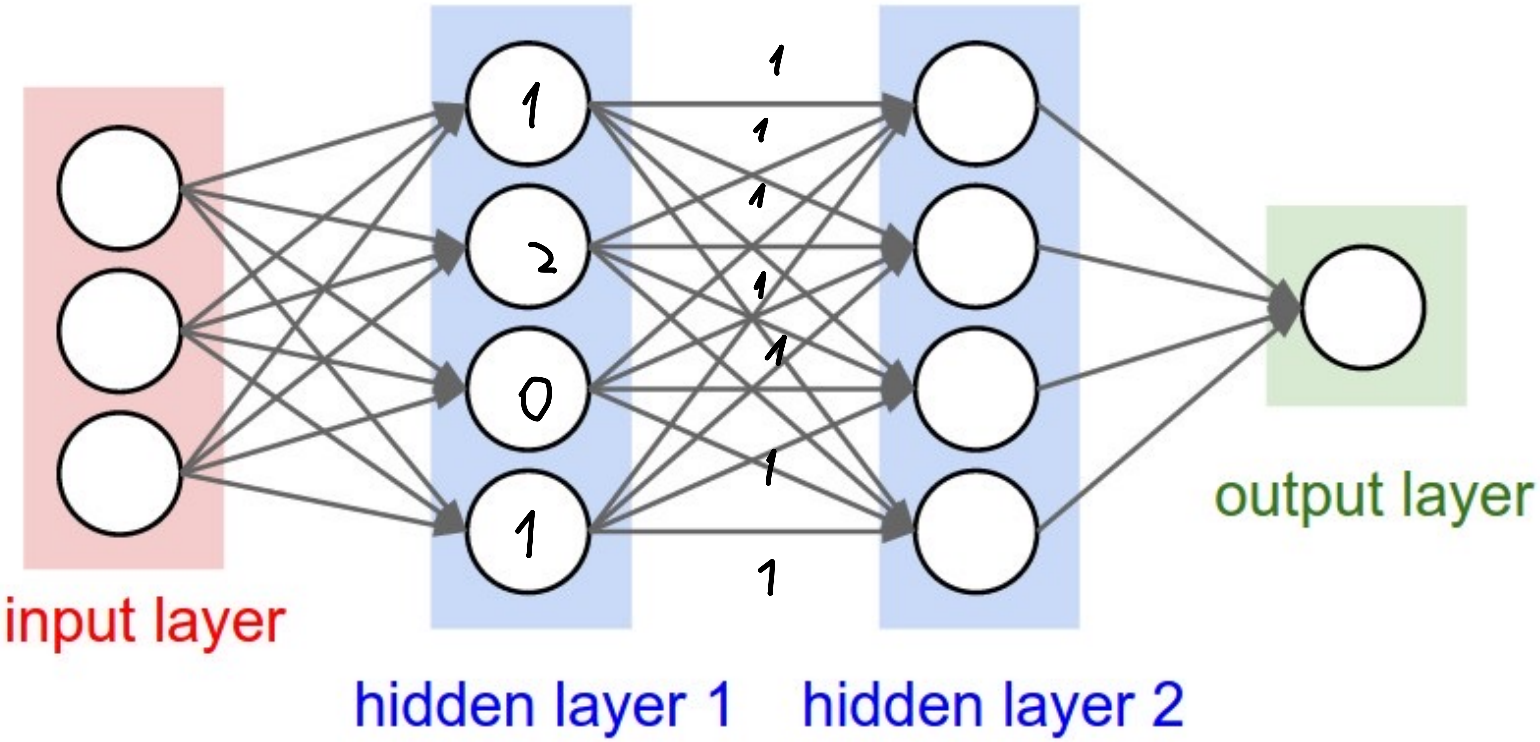
$$\sum_{i=1}^{\ell} s_i (a(x_i) - y_i)^2 \to \min$$

- Большие веса будем ставить объектам, которые похожи на объекты из тестовой выборки

Internal covariate shift

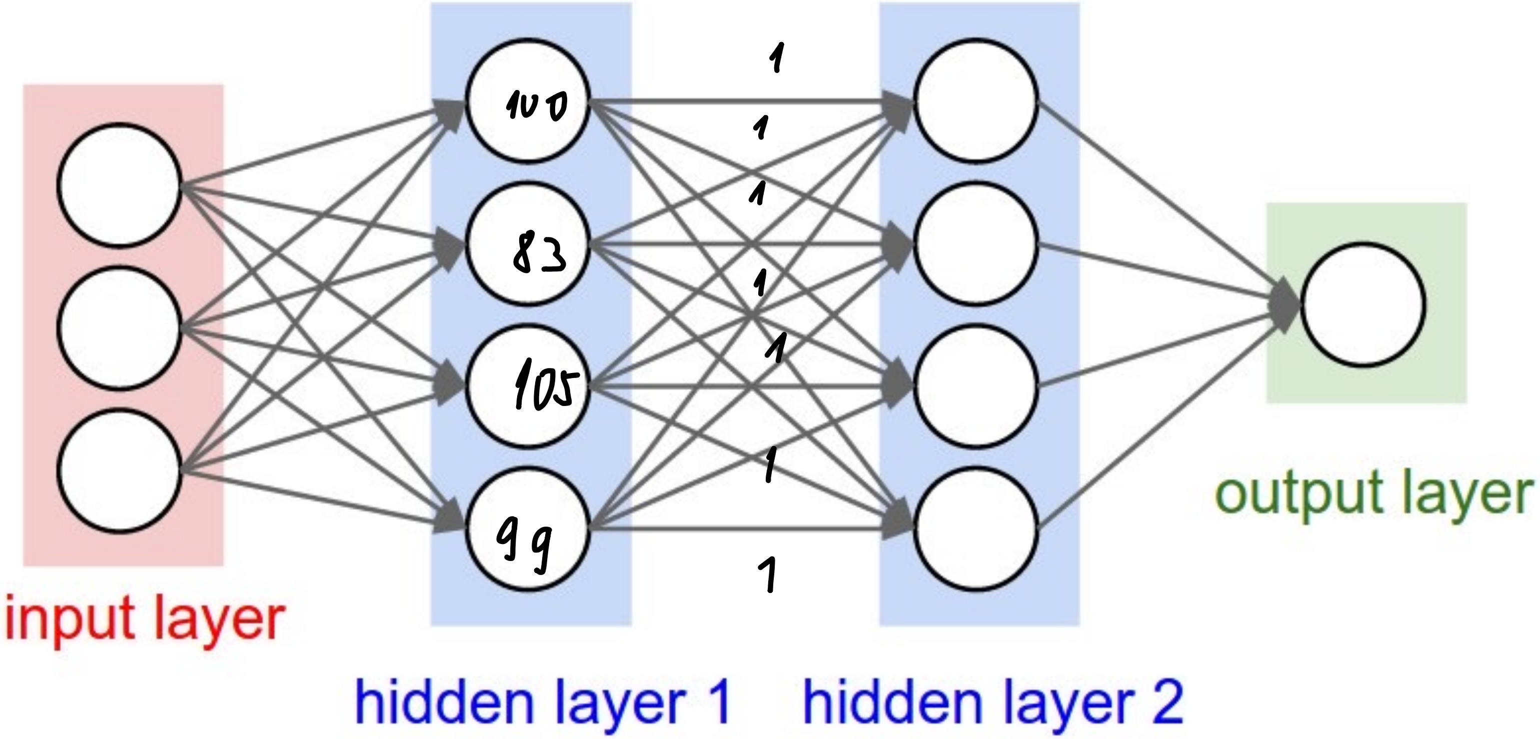
- В нейронной сети каждый слой обучается на выходах предыдущих слоёв
- Если слой в начале сильно меняется, то все следующие слои надо переделывать

Internal covariate shift



Internal covariate shift

Допустим, веса первого слоя сильно поменялись после градиентного шага



Internal covariate shift

- Идея: преобразовывать выходы слоёв так, чтобы они гарантированно имели фиксированное распределение

https://arxiv.org/pdf/1502.03167.pdf

Batch Normalization

- Реализуется как отдельный слой
- Вычисляется для текущего батча
- Оценим среднее и дисперсию каждой компоненты входного вектора:

$$\mu_B = \frac{1}{n}\sum_{j=1}^{n} x_{B,j}$$

покоординатно

$$\sigma_B^2 = \frac{1}{n}\sum_{j=1}^{n}\left(x_{B,j} - \mu_B\right)^2$$

$x_{B,j}$ — j-й объект в батче B

Batch Normalization

- Отмасштабируем все выходы:

$$\tilde{x}_{B,j} = \frac{x_{B,j} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

- Зададим нужные нам среднее и дисперсию:

$$z_{B,j} = \gamma \circ \tilde{x}_{B,j} + \beta$$

обучаемые параметры (векторы, размерность
равна размерности входных векторов)

Batch Normalization

$$\begin{pmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nd} \end{pmatrix}$$

Приводим среднее и дисперсию к β_1 и γ_1

Приводим среднее и дисперсию к β_d и γ_d

- n — размер батча
- d — размерность входного вектора

Batch Normalization

Важно: после BatchNorm среднее и дисперсия каждого выхода зависят только от параметров нормализации, но не от параметров прошлых слоёв!

Batch Normalization

Во время применения нейронной сети:

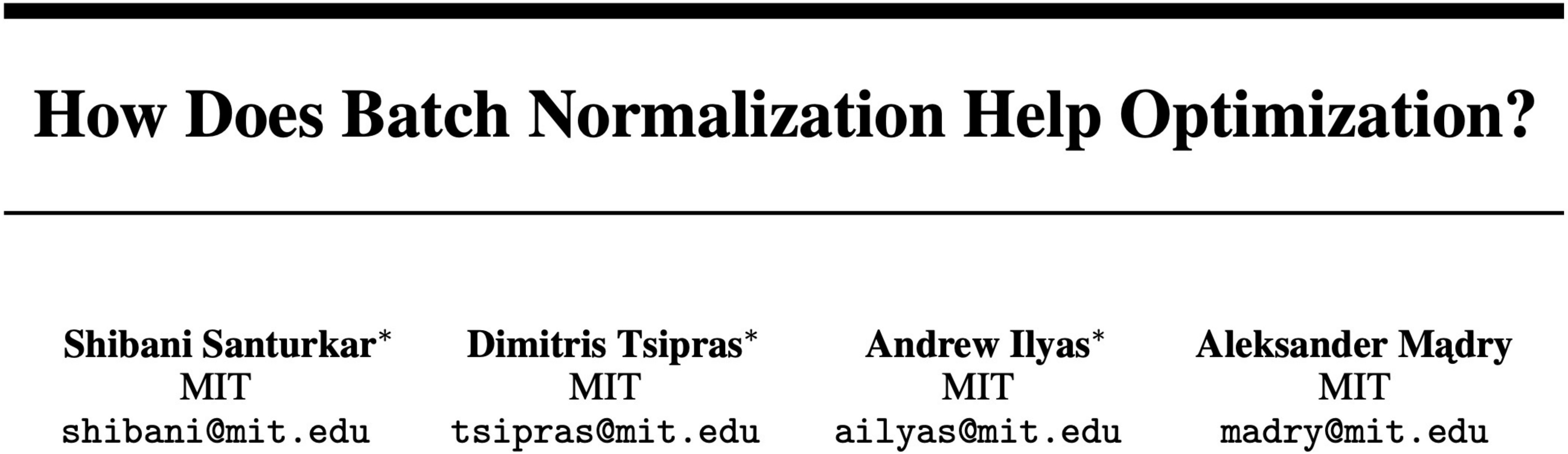
- Те же самые формулы, но вместо μ_B и σ_B^2 используем их средние значения по всем батчам

Batch Normalization

- Обычно вставляется между полносвязным/свёрточным слоём и нелинейностью
- Позволяет увеличить длину шага в градиентном спуске

- Не факт, что действительно устраняет covariance shift

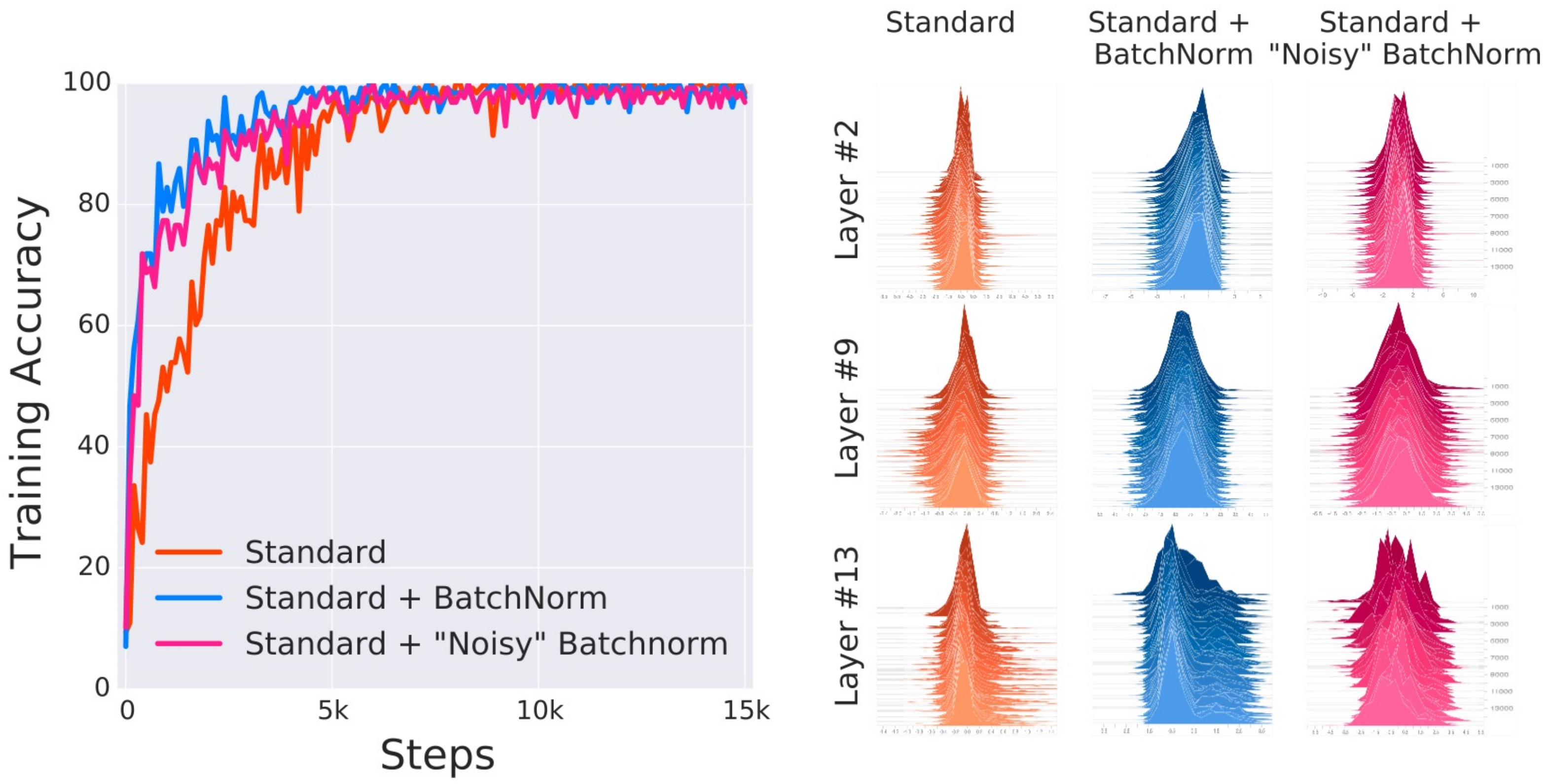
В чём польза от BatchNorm?



https://arxiv.org/pdf/1805.11604.pdf

В чём польза от BatchNorm?

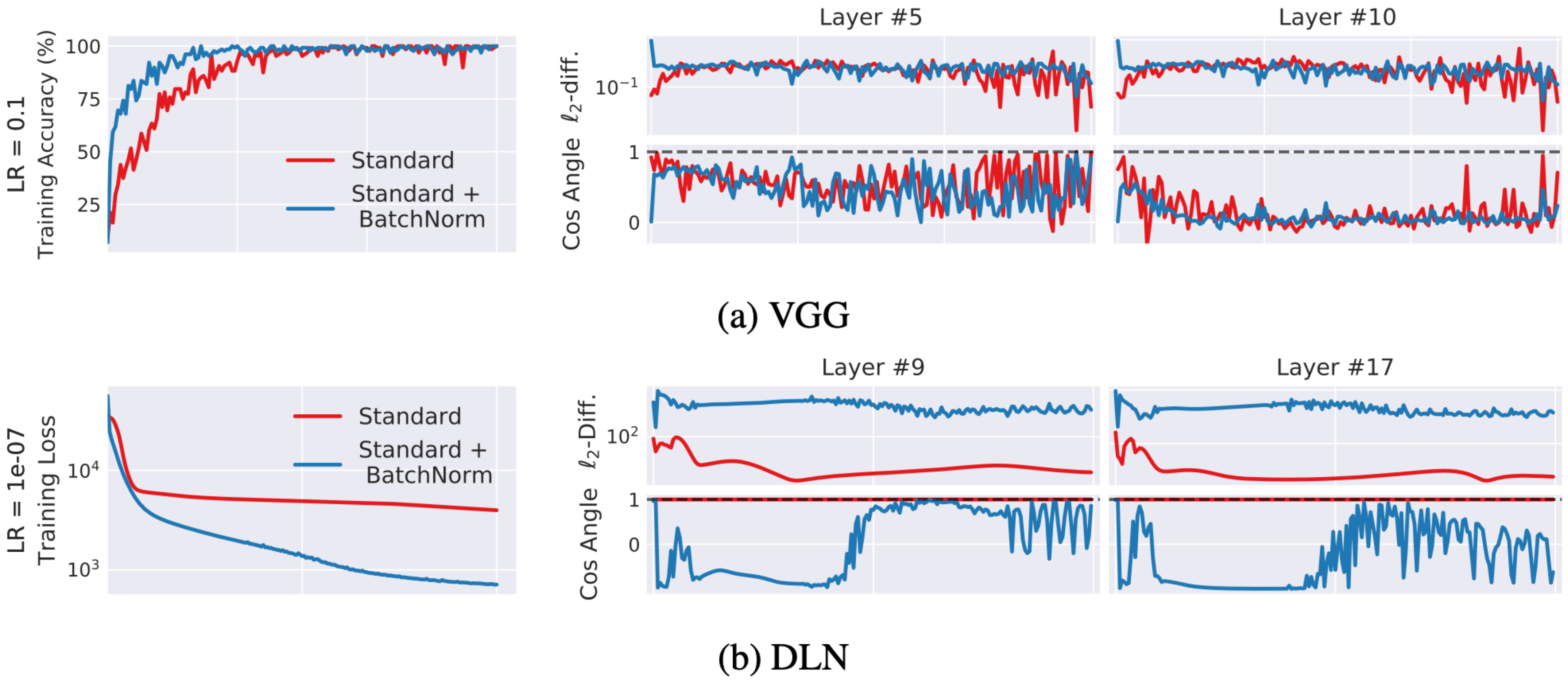
- Добавим шум после нормализации — хуже не становится!



https://arxiv.org/pdf/1805.11604.pdf

В чём польза от BatchNorm?

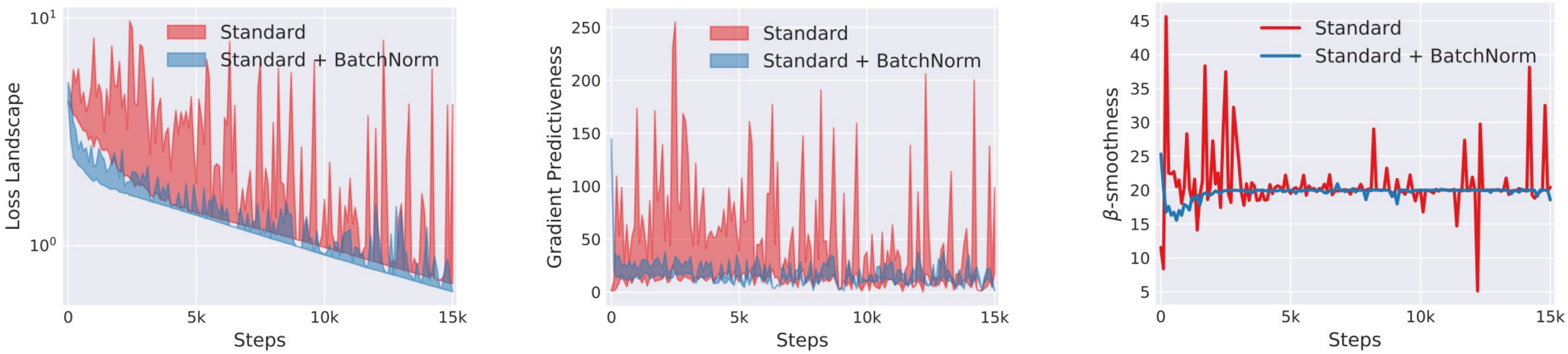
- Как связаны градиенты до и после обновления на предыдущих слоях?



https://arxiv.org/pdf/1805.11604.pdf

В чём польза от BatchNorm?

- Функционал ошибки становится более «гладким»!



(a) loss landscape (b) gradient predictiveness (c) "effective" β-smoothness

https://arxiv.org/pdf/1805.11604.pdf